Université de Tours
Licence 1 Informatique

Introduction à l'algorithmique

M1-3 Informatique 1 : Algorithmique — Semestre 1

1 Introduction

Définition 1.1. Un algorithme est une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre une classe de problèmes.

Exemple 1.2. On trouve des algorithmes un peu partout, par exemple :

- dans les recettes de cuisines
- algorithmes de compression audio vidéo (appel vidéo, visioconférence)
- calcul de la route la plus courte d'un point A vers un point B

Qu'est ce qu'un bon algorithme ?

Un bon algorithme doit (1) résoudre une classe de problèmes et (2) la résoudre de manière efficace. Ces deux considérations sont la base de l'algorithmique, nous reviendrons sur ces notions plus tard dans ce cours.

Pourquoi apprendre l'algorithmique pour apprendre à programmer ?

Parce que vous devez savoir programmer indépendamment des spécificités d'un langage. Apprendre l'algorithmique, c'est apprendre à manier la structure logique d'un programme informatique. Cette dimension est présente quelle que soit le langage de programmation.

Comment écrire un algorithme ?

Comme indiqué dans la définition, un algorithme doit être défini de manière **non ambigüe**. Pour cela, nous écrirons nos algorithmes en pseudo-code (qui sera défini plus loin dans ce chapitre). Les trois manière les plus courantes de décrire un algorithme sont

1. la langue courante
2. le pseudo-code
3. le langage de programmation.

La langue courante est la plus naturelle mais la moins précise, le langage de programmation est la plus précise mais la moins facile à comprendre. Le pseudo-code quand à lui se positionne quelque part au milieu, d'où son intérêt.

2 Valeurs et expressions

Les valeurs que nous allons manipuler peuvent être de différents types. Parmi les les types les plus courants on trouve :

- les entiers relatifs
- les réels
- les chaînes de caractères
- les booléens

Ces valeurs viennent naturellement avec des opérateurs. Un **opérateur** est un signe qui relie une ou plusieurs valeurs, pour produire un résultat. L'addition pour les entiers par exemple, qui à deux entiers associe un nouvel entier.

A l'aide des opérateurs et des valeurs, nous pouvons créer des expressions. Une **expression** est un ensemble de valeurs, reliées par des opérateurs, et équivalent à une seule valeur. En d'autre termes, lorsqu'on évalue une expression on doit trouver une seule valeur : par exemple $(2 + 3) + 2$ est une expression de valeur 7.

Remarque 2.1. Ce ne sont pas des définitions très rigoureuses d'opérateur ou d'expression mais cela nous suffira. C'est en fait assez compliqué (mais possible) de définir ces termes rigoureusement !

Le type entier et ses opérateurs

Les opérateurs classiques sur les entiers sont

- l'addition $+$, la soustraction $-$, la multiplication $\times$

- division entière `//` qui à deux entiers associe un entier : 7//2=3
- la division réelle `/` qui à deux entiers associe un réel : 7/2=3.5
- l'opérateur `mod` qui renvoie le reste de la division entière : 7 `mod` $2 = 1$, 11 `mod` $3 = 2$

Nous reviendrons en détails sur ces opérateurs plus tard dans le cours.

Le type chaîne de caractères et ses opérateurs

En pseudo-code, une chaîne de caractères est toujours notée entre guillemets ou entre quotes. L'opération prinicpale pour les chaînes de caractères est la concaténation que l'on notera `+`. Par exemple, si on concatène `'bon'` et `'jour'`, on obtient `'bonjour'`. On notera `'bon' + 'jour'` = `'bonjour'`

Remarque 2.2. On voit déjà apparaître une notion importante ici, l'opérateur `+` peut changer de sens selon que l'on applique à des entiers ou à des chaînes de caractères.

Le type booléen et ses opérateurs

Un booléen ne peut prendre que deux valeurs `Vrai` et `Faux`. Les opérateurs classiques sur les boléens sont :

- `et` qui est défini par la table de valeurs suivante où P et Q sont des valeurs boléennes :

P	Q	P et Q
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

- `ou` qui est défini par la table de valeurs suivante :

P	Q	P ou Q
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

⚠ Le `ou` en informatique n'est pas exclusif comme peut l'être parfois le `ou` que vous utilisez dans la vie courante. Pensez par exemple à la mention « fromage ou dessert » au restaurant.

- le `non` qui est défini par la table de valeur suivante :

P	`non` P
Vrai	Faux
Faux	Vrai

L'intérêt des valeurs booléennes réside notamment dans la possibilité de les produire en comparant des valeurs d'autres types, c'est ce que l'on appelle des **opérateurs de comparaisons** :

- test d'égalité $=$
- test d'inégalité $\neq$
- test de comparaison $<$
- test de comparaison $\geq$
- etc.

Par exemple , l'expression $1 = 2$ vaut `Faux`, l'expression $1 \neq 2$ vaut `Vrai`, l'expression $1 < 2$ vaut `Vrai` et l'expression $1 < 2$ `ou` $3 < 2$ vaut `Vrai`.

Exercice 1. Evaluer les expressions suivantes :

1. $1 > 2$ `et` $2 \leq 3$
2. $(1 > 2$ `et` $2 \leq 3)$ `ou` $1 = 1)$
3. `non` $\Big(\big(((2 \times 3) - 2) + 5//2) \leq ((7 \times 2) - 7)\big)$ `et` $\big(((6 - 3) \times 265) < ((6 - 2) \times 673)\big)\Big)$

3 Les variables

Pourquoi des variables ?

En algorithmique ou dans un programme informatique, on a besoin en permanence de stocker provisoirement des valeurs. Pour stocker une information au cours d'un programme, on utilise une variable.

Qu'est ce qu'une variable ?

Une variable est une boîte, que le programme va repérer par une étiquette (le **nom** de la variable). Pour avoir accès au contenu de la boîte (la **valeur**), il suffit de la désigner par son étiquette. Une variable possède un **type** qui caractérise l'ensemble des valeurs que peut prendre la variable (par exemple entiers, réels, chaînes de caractères etc.).

Comment créer une variable (en pseudo-code) ?

Il faut commencer par la déclarer, c'est-à-dire, par indiquer son nom et son type. La déclaration des variables se fait au début d'un algorithme avant la première instruction. En pseudo-code, on déclarera les variables sous la forme « nom : type ». Par exemple pour déclarer une variable entière intitulée a, on écrira « a : entier ». On peut déclarer plusieurs variables du même type par exemple en écrivant « a, b, c : entier ».

Comment donner une valeur à une variable (en pseudo-code) ?

Une fois la variable créée, on peut lui affecter une valeur correspondante à son **type**. En pseudo-code, l'affectation de variable se note avec le sigle $\leftarrow$. À gauche du signe $\leftarrow$ se trouve le nom de la variable à affecter, à droite du signe $\leftarrow$ se trouve une expression donnant la valeur à affecter à la variable.

Règles de bases concernant l'affectation

- A gauche de la flèche, il y a un nom de variable, et uniquement cela.
- une instruction d'affectation ne modifie que ce qui est situé à gauche de la flèche.
- L'ordre des instructions d'affectation est important !!

Exercice 2.

1. Quelles seront les valeurs des variables après exécution des instructions suivantes ?

a, b : entiers	a, b, c : entiers	ch1,ch2 : chaînes de caractères
$a \leftarrow 1$	$a \leftarrow 5$	ch1$\leftarrow$`'ba'`
$b \leftarrow a + 3$	$b \leftarrow 3$	ch2$\leftarrow$`'boum'`
$a \leftarrow 3$	$c \leftarrow a + b$	ch1$\leftarrow$ch1+ch1+ch2
	$a \leftarrow 2$	
	$c \leftarrow b - a$	

2. Écrire une suite d'instructions qui échange les valeurs de deux variables a et b.

4 Les algorithmes

4.1 Présentation

Dans la suite de ce cours, tous nos algorithmes seront décrits par une *entête* et un *corps*. Le corps de l'algorithme est délimité par les balises **Début** et **Fin**.

Présentation d'un algorithme

Nom : Le nom de l'algorithme
Rôle : Le rôle ou fonction de l'algorithme
Entrée : Les données en « entrée », c'est-à-dire les éléments indispensable au bon fonctionnement
Sortie : Les données en sortie, c'est-à-dire les éléments calculés, produits par l'algorithme
Déclaration : Les données locales de l'algorithme
Début
| Instruction 1
| Instruction 2
| ⋮
| Instruction n
Fin

Exemple 4.1. Première exemple : l'additon de deux entiers.

Nom : Addition_deux_entiers
Rôle : Retourne la somme de deux entiers a et b
Entrée : a, b : entier
Sortie : c : entier
Déclaration : -
Début
  $c \leftarrow a + b$
  Renvoyer c
Fin

4.2 Lecture et écriture

Il existe des d'instructions pour permettre à la machine de dialoguer avec l'utilisateur. Dans un sens, ces instructions permettent à l'utilisateur de rentrer des valeurs pour qu'elles soient utilisées par le programme. Cette opération est la **lecture**. Dans l'autre sens, d'autres instructions permettent au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran. Cette opération est l'**écriture**.

Pseudo-code pour la lecture

Pour que l'utilisateur entre la (nouvelle) valeur de la variable a, on mettra simplement « Lire(a) ». Cette variable doit bien entendu avoir été déclarée au début de l'algorithme.

Pseudo-code pour l'écriture

Pour écrire le contenu d'une variable ou d'une valeur, on mettra simplement « Écrire » suivi du nom de la variable ou la valeur entre parenthèse.

Remarque 4.2. Il est **très fortement conseillé** de commencer par écrire des informations avant de demander à l'utilisateur d'entrer une valeur ! Par exemple :

```
Écrire('Entrez votre nom')
Lire(nom_famille)
```

A l'aide de ces deux commandes, on peut écrire des algorithmes en demandant la valeur des variables à l'utilisateur au lieu de les avoir en entrée. On peut aussi afficher le résultat plutôt que de le renvoyer... La différence est subtile mais importante, nous en reparlerons souvent dans ce cours.

Exemple 4.3. L'addition de deux entier avec dialogue utilisateur (pas d'entrée, pas de sortie). Comparer avec l'Exemple 4.1.

Nom : Addition_deux_entiers
Rôle : Affiche la somme de deux entiers a et b donnés par l'utilisateur
Entrée : -
Sortie : -
Déclaration : a, b, c : entier
Début
  Écrire('Entrez le premier entier ')
  Lire(a)
  Écrire('Entrez le deuxième entier')
  Lire(b)
  $c \leftarrow a + b$
  Écrire('La somme des entiers a et b vaut',c)
Fin

4.3 Les expressions conditionnelles

L'instruction **Si alors sinon** permet de conditionner l'exécution d'une instruction à la valeur d'une expression booléenne (qui vaut, on le rappelle, `Vrai` ou `Faux`). Sa forme la plus simple est :

si `condition` **alors**
| instruction1
fin

La structure d'un test est relativement claire. Arrivée à la première ligne, on évalue le booléen `condition`, si celui-ci est `Vrai` alors on execute l'instruction 1, sinon on ne fait rien.

On peut rajouter une condition sinon :

si `condition` **alors**
| instruction1
sinon
| instruction2
fin

Encore une fois, la structure est relativement claire. Arrivé à la première ligne, on évalue le booléen `condition`, si celui-ci est vrai alors on exécute l'instruction 1, sinon on exécute l'instruction 2.

Exemple 4.4. On considère les instructions suivantes :

$a \leftarrow 2$	$a \leftarrow 1$
si $a < 2$ **alors**	**si** $a < 2$ **alors**
$\quad a \leftarrow 2 \times a$	$\quad a \leftarrow 2 \times a$
sinon	**sinon**
$\quad a \leftarrow a - 1$	$\quad a \leftarrow a - 1$
fin	**fin**

Dans l'exemple de gauche, à la fin des instructions la variable a vaut 1. Dans l'exemple de droite, à la fin des instructions la variable a vaut 2.

Finalement on peut encore faire :

si `condition1` **alors**
| instruction1
sinon si `condition2` **alors**
| instruction2
sinon
| instruction3
fin

Attention, ici une et une seule des instructions est exécutée. Si les boléens `condition1` et `condition1` valent `Vrai`, seule l'instruction 1 sera executée...

Exemple 4.5. Nous allons écrire un algorithme qui permet de calculer de la valeur absolue d'un réel

Nom : Abs
Rôle : Retourne la valeur absolue d'un réel
Entrée : a : réel
Sortie : abs : réel
Déclaration : -
Début
| **si** $a \geq 0$ **alors**
| $\quad abs \leftarrow a$
| **sinon**
| $\quad abs \leftarrow -a$
| **fin**
| **Renvoyer** abs
Fin

Dans les 3 exercicies suivants, on écrira deux versions de l'algorithme : une dans laquelle les entiers sont données en entré de l'algorithme, l'autre avec dialogue utilisateur (voir Exemples 4.1 et 4.3)

Exercice 3. Écrire un algorithme qui détermine le maximum (le plus grand) de deux entiers.

Exercice 4. Écrire un algorithme qui détermine si un entier est pair ou impair.

Exercice 5. Écrire un algorithme qui détermine le maximum (le plus grand) de trois entiers.

4.4 Les boucles Tant que

Elle apparaît sous la forme suivante :

tant que `condition` **faire**
| instruction
fin

En d'autres termes, tant que le boléen `condition` est `Vrai`, on exécute l'instruction.

Exemple 4.6. On considère la suite d'instruction suivante :

$a \leftarrow 0$
$i \leftarrow 1$
tant que $i \leq 5$ **faire**
| $a \leftarrow a + i$
| $i \leftarrow i + 1$
fin

La variable a vaut alors 15.

Exemple 4.7. L'algorithme suivant calcule la somme des carrés des n premiers entiers.

Nom : Somme_des_carres
Rôle : Renvoie la somme des carrés des n premiers entiers
Entrée : n : entier
Sortie : *somme* : entier
Déclaration : -
Début
| $somme \leftarrow 0$
| $i \leftarrow 0$
| **tant que** $i \leq n$ **faire**
| | $somme \leftarrow somme + i \times i$
| | $i \leftarrow i + 1$
| **fin**
| **Renvoyer** somme
Fin

Exercice 6. Écrire la trace (c'est-à-dire l'évolution des différentes variables au fur et à mesure de l'exécution) de l'algorithme suivant :

Nom : Etrange
Rôle : mystère
Entrée : -
Sortie : -
Déclaration : a, b : entier
Début
| $a \leftarrow 3$
| $b \leftarrow 96$
| **tant que** $b \geq 2$ **faire**
| | $b \leftarrow b$ // a
| | $a \leftarrow a + 2$
| **fin**
Fin

Exercice 7. Écrire un algorithme qui calcule la somme de l'inverse des n premiers entiers.